Framework: The Prototype Project

Gordon Watts
Scott Snyder
9/16/97
Framework Group

- What is a framework?
- What did we do?
- How well did it work?

What!?

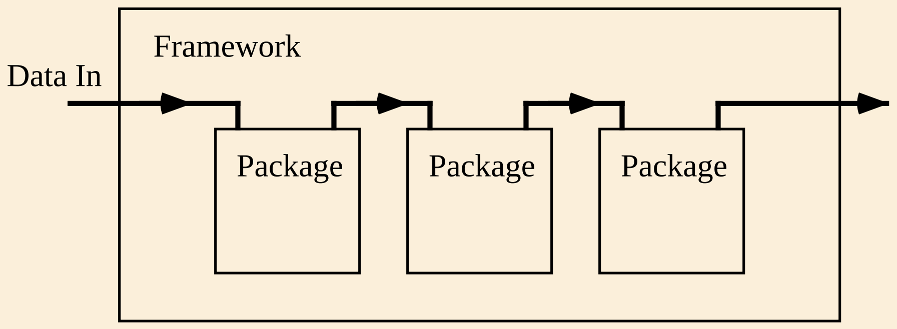
- Not just the production environment
  - Future physicst's introduction to D∅ software.
  - Analysis programming
  - ... and farm programming.
- Basic concepts
  - Isolate physics code from data management, calibration, and other often repeated steps
  - Supply a UI for both batch and user environments
  - Allow code to move from one environment to another unchanged.
  - Error recovery and management

Framework Environments

- Environments
  - Farm
    - No GUI
    - Careful accounting for data done
  - Analysis Batch Jobs
    - Similar to farm environment
  - Analysis / Debugging
    - GUI
    - Record actions for use in batch job
    - Must work with debugger
  - Online Machines (not Level 3)
    - Unique GUI requiremnts (think global monitor from last run).
  - Trigger Filters
    - Too late already; another interface has already been designed. Perhaps different interface required anyway.
    - Will write interface to the framework for trigger algorithm development, however.
- Other things to be aware of:
  - Does the framework handle SMP systems by running multithreaded?

The Framework Model

- Can split the domain of an analysis program in two:
  - Hardcore physics analysis (package)
    - data access
    - complex algorithms (CPU time)
  - Coordination (framework)
    - data file i/o
    - calibration
    - data routing (getting data to physics algorithms)
    - gui/batch interface translation
- Similar to the Run I model:



The Framework Model

- Make the event routing flexible
  - Don’t limit self to assembly line model.
- Make the framework code itself as ignorant of data anlysis as possible, put everything in packages.
  - Event I/O.
  - Calibration and Geometry data I/O processing.
- Almost all communication between packages is through the event model.
- This model is based on data flow
  - The data is the event.

Scripting

- How to implement data flow part of framework?
  - Keep it flexable
  - Allow for dynamic loading of modules
- Other considerations
  - RCP input format was not well liked
    - Headache keeping track of RCP file when test release released new version, but you didn't relink
    - RCP files were dumb; couldn't pick up info from command environment when running lots of batch jobs each with a minor modification to an RCP.
  - Unify the various input formats
    - RCP, program builder, etc.
  - GUI
    - Programming directly to X/Motif is _very_ ugly.

Scripting

- A scripting language can solve much of this
  - Search performed; Python is winner.
- Python
  - cross platform GUI interface
  - access C++ objects from Python and vice versa.
  - Comfortable embedding in other programs
  - Dynammic linking of modules built in for various platforms
- TCL
  - CDF choose this guy.
  - Supported by CD, comes in stripped down form
    - So does python, but we had to do it.
  - Known in other parts of the lab.
- I still think Python is the right choice.

Liz Sexton Says...

Scripting

- Pro
  - Potential to solve many of the problems/other considerations listed
    - RCP input
    - GUI
    - Powerful enough to do other input forms.
  - Supported by large user community (not us)
  - Flexible event routing (including multievent input packages?) could be handled.
- Cons
  - Increases program size (bloat)
  - Programs are now written in two different languages; debugging across the boundries won’t be easy.
  - How much do users have to learn?

Components

- Simple assembly line prototype consists of many of the same components as Run I.
  - Program Builder
    - Puts together outline, generates initial RCP files that specify packages to include
    - Generates makefile (with link list).
    - Include defaut packages for event i/o, etc.
  - Packages
    - Plain interface to accept event data.
    - I/O as well as physics analysis
    - Could be written in C++ or Python.
    - Superpackage could contain other packages, and make routing decisions.
  - Framework
    - Master event loop
    - Builds package list
    - Runs data through the list.

Where to find Info

- Framework Homepage:
  - http://www-d0.fnal.gov/ → Computing → Offline → Frameworks.
- Online
  - setup D0RunII
  - look at the framework package.
- Examples/Tutorials
  - http://www-d0.fnal.gov/~gwatts/upgrade_software/frame/Welcome.html

What is done?

- Samples and Tutorials are up and running
  - Can't use because I've not added the code to CVS to automatically define commands when you do a setup.
- RCP Stuff
  - Reads old format
  - New format that is a python script like format.
- sample modules
  - Some in C++, some in Python
- Interface classes between Python and C++
  - Makes interface nicer than using direct Python calls.
  - Can handle almost any scripting language that has OO in it in some fasion.

Not?

- GUI
  - How we might do interface to packages
    - Use callbacks to define python scripts on a per package basis. Too complex?
  - How would we do RCP parameters?
- No real program control
- Understand the different types of events
  - based on data flow.
  - base on dataflow for _all_ types?
- Threading or other forms of processing multiple events
- Solve the RCP proliferation problem
  - Keep a master RCP database along with the built code.?
- Multi-event input to package

How well does it work?

- Complexity
  - Still need a program builder
  - Link libraries are still a pain.
  - Anyway to make it _simpler_?
- Still too many files left in a user’s directory for my taste.

Other experiments

- read mail, buddy!

Conclusions

- None. This is the first meeting of a new group!
- Not enough sleep to think straight.